

EJERCICIO 11

ENUNCIADO

Ejercicio 11 Considere el siguiente programa escrito en *assembler* de la arquitectura ORGA1:

```

Vector: DW ...
        DW ...
        ...
        DW 0000
Clave:  DW ...
Comienzo: MOV R0, Vector
        MOV R1, [Clave]
        MOV R2, 0
Ciclo:  MOV R3, [R0]
        CMP R3, 0
        JE Fin
        CMP R3, R1
        JE Sumo
Sigo:   ADD R0, 1
        JMP Ciclo
Sumo:   ADD R2, 1
        JMP Sigo
Fin:    RET

```

- El programa se ubica en la posición 0x0100 de la memoria. Los vectores se almacenan como valores contiguos de una palabra de tamaño. Para indicar la finalización del vector se utiliza el valor cero. Suponiendo que el vector contiene diez palabras, indicar el valor de cada una de las etiquetas.
- Explicar qué hace el programa siguiendo su ejecución paso a paso.
- Escribir un *pseudocódigo* que refleje el comportamiento del programa.
- Mostrar un vector para el cual este programa no hace lo esperado.

INCISO A

Para calcular el valor de las etiquetas, veo el tamaño de las instrucciones del programa (teniendo en cuenta que el vector contiene diez palabras):

Vector:	DW ...	→ 1 palabra
	DW ...	→ 1 palabra
	...	
	DW 0000	→ 1 palabra
Clave:	DW ...	→ 1 palabra
Comienzo:	MOV R0, Vector	→ 2 palabras
	MOV R1, [Clave]	→ 2 palabras
	MOV R2 0x0000	→ 2 palabras
Ciclo:	MOV R3, [R0]	→ 1 palabra
	CMP R3, 0x0000	→ 2 palabras
	JE Fin	→ 1 palabra
	CMP R3, R1	→ 1 palabra
	JE Sumo	→ 1 palabra
Sigo:	ADD R0, 0x0001	→ 2 palabras
	JMP Ciclo	→ 2 palabras
Sumo:	ADD R2, 0x0001	→ 2 palabras
	JMP Sigo	→ 2 palabras
Fin:	RET	→ 1 palabra

Ahora, sabiendo que el programa se carga en la dirección de memoria 0x0100, calculo:

- Vector = 0x0100 + 0x0000 = 0x0100
- Clave = 0x0100 + 0x000A (10 palabras del vector) + 0x0001 = 0x010B
- Comienzo = 0x0100 + 0x000A + 0x0002 = 0x010C
- Ciclo = 0x0100 + 0x000A + 0x0002 + 0x0006 = 0x0100 + 0x0012 = 0x0112
- Sigo = 0x0100 + 0x000A + 0x0002 + 0x0006 + 0x0006 = 0x0100 + 0x0018 = 0x0118
- Sumo = 0x0100 + 0x000A + 0x0002 + 0x0006 + 0x0006 + 0x0004 = 0x0100 + 0x001C = 0x011C
- Fin = 0x0100 + 0x000A + 0x0002 + 0x0006 + 0x0006 + 0x0004 + 0x0004 = 0x0100 + 0x0020 = 0x0120

INCISO B

Seguimos la ejecución del programa teniendo en cuenta que PC comienza en **Comienzo**, o sea, $PC = 0x010C$. Calculamos luego los valores de los desplazamientos de todas las instrucciones de salto condicional:

- JE Fin: Dir: 0x0115; Desplazamiento = $0x0120 - (0x0115 + 0x0001) = 0x0120 - 0x0116 = 0x0A$ (en 8b)
- JE Sumo: Dir: 0x0117; Desplazamiento = $0x011C - (0x0117 + 0x0001) = 0x011C - 0x0118 = 0x04$ (en 8b)

Con esto, pasamos el programa a lenguaje máquina (desde la etiqueta **Comienzo** puesto que el vector posee valores genéricos y la clase también):

Comienzo:	MOV R0, Vector	→ Bin: 0001 1000 0000 0000 0000 0001 0000 0000	Hex: 1800 0100
	MOV R1, [Clave]	→ Bin: 0001 1000 0100 1000 0000 0001 0000 1011	Hex: 1848 010B
	MOV R2 0x0000	→ Bin: 0001 1000 1000 0000 0000 0000 0000 0000	Hex: 1800 0000
Ciclo:	MOV R3, [R0]	→ Bin: 0001 1000 1111 0000	Hex: 18F0
	CMP R3, 0x0000	→ Bin: 0110 1000 1100 0000 0000 0000 0000 0000	Hex: 68C0 0000
	JE Fin	→ Bin: 1111 0001 0000 1010	Hex: F10A
	CMP R3, R1	→ Bin: 0110 1000 1110 0001	Hex: 68E1
	JE Sumo	→ Bin: 1111 0001 0000 0100	Hex: F104
Sigo:	ADD R0, 0x0001	→ Bin: 0010 1000 0000 0000 0000 0000 0000 0001	Hex: 2800 0001
	JMP Ciclo	→ Bin: 1010 0000 0000 0000 0000 0001 0001 0010	Hex: A000 0112
Sumo:	ADD R2, 0x0001	→ Bin: 0010 1000 1000 0000 0000 0000 0000 0001	Hex: 2880 0001
	JMP Sigo	→ Bin: 1010 0000 0000 0000 0000 0001 0001 1000	Hex: A000 0118
Fin:	RET	→ Bin: 1100 0000 0000 0000	Hex: C000

Ahora, vemos en base a esto que el programa comienza moviendo a R0 la **dirección de memoria** en la que está cargada el vector, a R1 el valor de la **celda** etiquetada como clave y a R2 le pasa 0. Luego, mueve a R3 el valor de la celda actual del vector y se fija si es 0 para terminar la ejecución de la subrutina en RET. En caso contrario, compara el valor de la celda del vector (R3) con la clave (R1) y en caso de ser iguales aumenta el valor de R2 en 1. En cualquier caso, seguido a esto R0 alberga la dirección de memoria consecutiva a la celda sobre la que se está operando (R3) y se repite el proceso renovando el valor de R3 con el de la nueva celda.

Básicamente, el programa **itera** sobre las celdas del vector con R0, guarda la clave en R1, almacena en R3 el valor de cada palabra del vector y en R2 mantiene un **contador** de la cantidad de palabras del vector iguales a la clave.

INCISO C

En base a la explicación anterior, escribimos el pseudocódigo del programa como:

Algoritmo 1: Pseudocódigo del programa

```

1 i ← 0
2 iguales ← 0
3 while i < len(vector) do
4   | if vector[i] == clave then iguales++;
5   | i++;
6 end
7 return
```

Aquí tomamos a **vector** como un arreglo de enteros, al registro R0 como el iterador **i** de posiciones, al registro R1 como **clave** y al registro R2 como **iguales**.

INCISO D

Un caso en que dicho programa puede no hacer lo esperado es cuando tenemos un vector de al menos $0x8000 = 2^{15}$ palabras **iguales a la clave**. En dicho caso, el contador (R3) aumentará en 1 por cada una de estas palabras en el vector. Luego de haber recorrido $0x7FFF$ palabras iguales a la clave, **R3 = 0x7FFF**.

Ahora, en la siguiente iteración con una palabra igual la clave (queda al menos una), se ejecutará $ADD\ R3,\ 0x0001 \rightarrow R3 = 0x7FFF + 0x0001 = 0x8000$ y los flags nos indicarán que el resultado es **negativo** (su cifra más significativa es 1 al representarse en complemento a 2), lo cual no coincide con la cantidad buscada, la cual es **mayor a 0**.

Nótese que el vector no podrá tener un tamaño **demasiado grande** de palabras, debido a que la memoria tiene 65536 celda de memoria utilizables y parte de ellas están siendo usadas por el programa.